A

Project Report

ON

Self-Driving Car System

In partial fulfillment of requirements for the degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

Submitted by:

Vinayak Lal (2017BTCS106)

Udit Tiwari (2017BTCS102)

Meenal Srivastava (2017BTCS084)

Ashwini Joshi (2017BTCS054)

SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

SYMBIOSIS UNIVERSITY OF APPLIED SCIENCES, INDORE

2017-2021

SYMBIOSIS UNIVERSITY OF APPLIED SCIENCES, INDORE

SCHOOL OF COMPUTER SCIECNE AND INFORMATION TECHNOLOGY

DECLARATION

We here declare that work which is being presented in the project entitled **"Self-Driving car System"** in partial fulfillment of degree of **Bachelor of Technology in computer science and Information Technology, Indore** is an authentic record of our work carried out under the supervision and guidance of **Dr. Neha Gupta** Asst. Professor of computer science and Information Technology. The matter embodied in this project has not been submitted for the award of any other degree.

Vinayak Lal

Udit Tiwari

Meenal Srivastava

Ashwini Joshi

Date :

SYMBIOSIS UNIVERSITY OF APPLIED SCIENCES, INDORE
SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

<u>PROJECT APPROVAL SHEET</u>

Following team has done the appropriate work related to the **"Self-Driving Car System"** in partial fulfillment for the award of **Bachelor of Technology in computer science and Information Technology** of "Symbiosis University of Applied Sciences, Indore".

Team:
1. **Vinayak Lal**
2. **Udit Tiwari**
3. **Meenal Srivastava**
4. **Ashwini Joshi**

Internal Examiner **External Examiner**

Date

SYMBIOSIS UNIVERSITY OF APPLIED SCIENCES, INDORE
SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that **Mr. Vinayak Lal, Mr. Udit Tiwari, Ms. Meenal Srivastava and Ms. Ashwini Joshi** working in a team have satisfactorily completed the project entitled **“Self-Driving Car System”** under my guidance in the partial fulfillment of the degree of **Bachelor of Technology in computer science and Information Technology** awarded by Symbiosis University of Applied Sciences Indore during the academic year 2020-2021

Dr.Neha Gupta
Project Coordinator

ACKNOWLEDGMENT

We are grateful to a number of persons for their advice and support during the time of complete our project work. First and foremost, our thanks goes to Director of the Department of Computer science and Information Technology and the mentors of our project for providing us valuable support and necessary help whenever required and also helping us explore new technologies by the help of their technical expertise. Their direction, supervision and constructive criticism were indeed the source of inspiration for us.

We are also thankful to **Dr.Neha Gupta,** project coordinator for helping us in rendering administrative support during the project work. We also owe our sincere thanks to all the **faculty members** of computer science and Information Technology Department who have always been helpful in providing guidance from time to time.

We forward our sincere thanks to all **teaching and non-teaching staff** of computer science and Information Technology department, SUAS Indore for providing necessary information and their kind co-operation.

We would like to thanks our parents and family members, our classmates and our friends for their motivation and there valuable suggestion during the project. Last, but not the least, we thank all those people, who have helped us directly or indirectly in accomplishing this work. It has been a privilege to study at Symbiosis University of Applied Sciences Indore.

ABSTRACT

1.25 million people die in road accidents globally every year, or on an average 3287 deaths per day and 20-50 million people get injured and disabled for life annually, as reported by World Health Organization (WHO). International Road Federation (IRF), Geneva report says that India has the highest number of road deaths and ranking first while accounting for 10% of global road accidents. It is found that 78% of road accidents are due to drivers' fault or human error in driving and it is interesting to address this human issue with deployment of technology in the form of Driverless Car. This helps in significant savings in time and will reduce the number of road accidents.

A self-driving car is a car that can sense its surrounding and navigate without any human input. Self-driving cars are also known as autonomous cars, driver less cars, or robotic cars. Currently, cars are semi-automated and require human control.

Autonomous cars work with the use of different technologies including laser light, GPS, computer vision, radar, etc.

There are many challenges in the field and research is going in full-flow from the best researchers in the industry.

TABLE OF CONTENT

Chapter 1

INTROCUTION

1.1 Problem Statement

1.25 million people die in road accidents globally every year, or on an average 3287 deaths per day and 20-50 million people get injured and disabled for life annually, as reported by World Health Organization (WHO). International Road Federation (IRF), Geneva report says that India has the highest number of road deaths and ranking first while accounting for 10% of global road accidents. In this project a low-cost prototype of self-driving car is proposed and implemented. The car will have a camera on board and with the feed video the analyzer computer can detect traffic signal (turn right, turn left, stop) and give correct decisions to the car. It is found that 78% of road accidents are due to drivers' fault or human error in driving and it is interesting to address this human issue with deployment of technology in the form of Driverless Car. This helps in significant savings in time and will reduce the number of road accidents.

1.2 Existing System

This 21st-century gold rush is motivated by the intertwined forces of opportunity and survival instinct. By one account, driverless tech will add $7 trillion to the global economy and save hundreds of thousands of lives in the next few decades. Simultaneously, it could devastate the auto industry and its associated gas stations, drive-thru, taxi drivers, and truckers. Some people will prosper. Most will benefit. Many will be left behind. It's worth remembering that when automobiles first started rumbling down manure-clogged streets, people called them horseless carriages. The moniker made sense:

Here were vehicles that did what carriages did, minus the hooves. By the time “car” caught on as a term, the invention had become something entirely new. Over a century, it reshaped how humanity moves and thus how (and where and with whom) humanity lives. This cycle has restarted, and the term “driverless car” will soon seem as anachronistic as “horseless carriage.” We don’t know how cars that don’t need human chauffeurs will mold society, but we can be sure a similar gear shift is on the way.

1.3 Literature Survey

The purpose of this paper is to conduct a literature survey on the research of self-driving car which operating in an urban environment. The chosen literature is the literature published since the 2007 DARPA Urban Challenge (DUC) competition and fulfills the criteria of autonomous vehicle level 3 or higher based on the Society of Automotive Engineers (SAE). In addition to using sources from scientific publications, this survey also carried out through websites of companies that develop self-driving car as well as through news published on various media. This is because not all commercial companies publish the results of their research through scientific papers. The results of this survey are a comparison of the main technologies used in the existing self-driving car systems, as well as comparing the advantages and disadvantages of each of these technologies. The conclusion can help researchers find the latest information about self-driving car technology and find a direction of research in the field of self-driving car.

1.4 Proposed System

The impending introduction of autonomous vehicles (AVs) has posed regulatory and ethical questions regarding how they should operate. Much of the previous literature on this subject has explored these questions with an underlying model of streets based on the present.

This project takes a different approach by putting forward a future vision for streets where privately owned and operated vehicles are no longer dominant and shared transit is more pervasive. In doing so, this work expands the current discussions around individual AVs to the system of streets they will occupy. It views the topology of streets and the rules that govern them, coupled with the vehicles that move through the streets, as an autonomous system, or machine. This project proposes updates to this autonomous system in order to build a more equitable system for a future where AVs will be ubiquitous. We present a design with two parts in order to ensure that AVs operate in the public's best interests:

1. An update to the laws that govern the use of roads, vehicle regulations and safety standards.
2. A requirement that AV decision making code be open sourced.

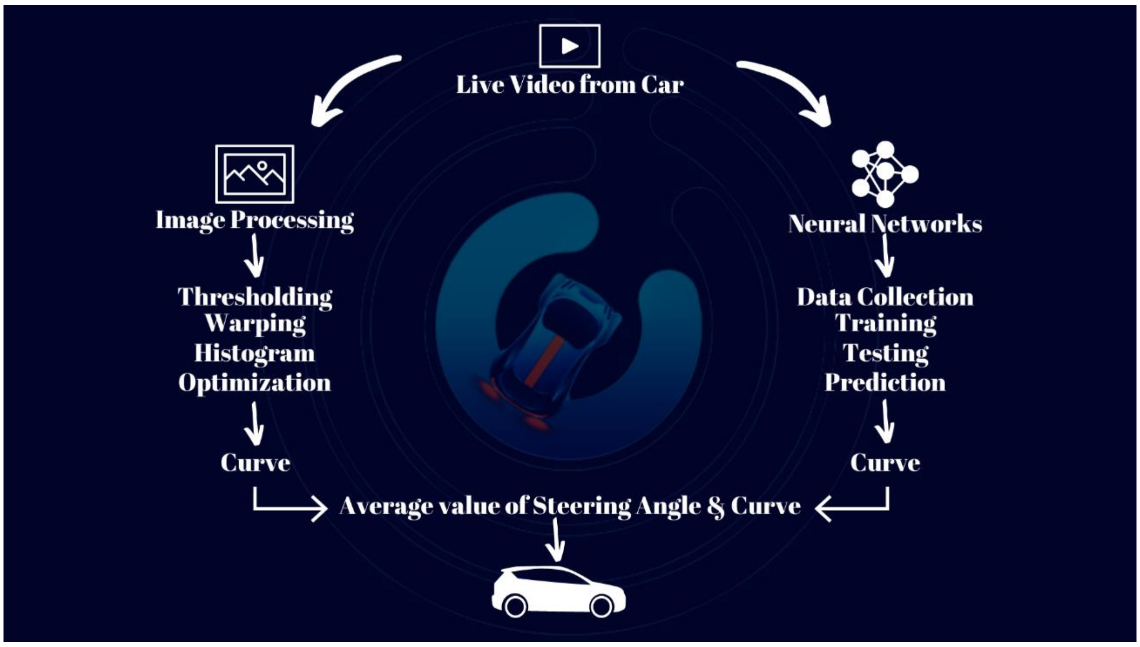


Fig.1- Block Diagram of Proposed System

1.5 Feasibility Study

Before starting the project, feasibility study is carried out to measure the viable of the system. Feasibility study is necessary to determine if creating a new or improved system is friendly with the cost, benefits, operation, technology and time. Following feasibility study is given as below:

Technical Feasibility

Technical feasibility is one of the first studies that must be conducted after the project has been identified. Technical feasibility study includes the hardware and software devices. ;

- **Software**:

1. Python
2. Linux OS
3. MU Code

- **Raspberry Pi System**:

1. Raspberry Pi 4 Model
2. Raspberry Pi Camera Module

- **Motor Control System**

1. DC Geared Motors
2. L298N Motor Driver

1.5.2. Operational Feasibility

Operational Feasibility is a measure of how well a proposed system solves the problem and takes advantage of the opportunities identified during scope definition. The following points were considered for the project’s technical feasibility:

- Correct Sensor Installation.
- Transfer Data to Cloud
- Machine Learning Models Validation

Modules Of System

Proposed system consist of following sensors and modules:

1. Raspberry Pie

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It’s capable of doing everything you’d expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games. The Raspberry Pi is a very cheap computer that runs Linux, but it also provides a set of GPIO (general purpose input/output) pins that allow you to control electronic components for physical computing and explore the Internet of Things (IoT).

Fig 1 Raspberry Pi

2. Ultrasonic sensor

An ultrasonic sensor is an electronic device that measures the distance of a target object by emitting ultrasonic sound waves, and converts the reflected sound into an electrical signal. Ultrasonic waves travel faster than the speed of audible sound (i.e. the sound that humans can hear). Ultrasonic sensors have two main components: the transmitter (which emits the sound using piezoelectric crystals) and the receiver (which encounters the sound after it has travelled to and from the target).

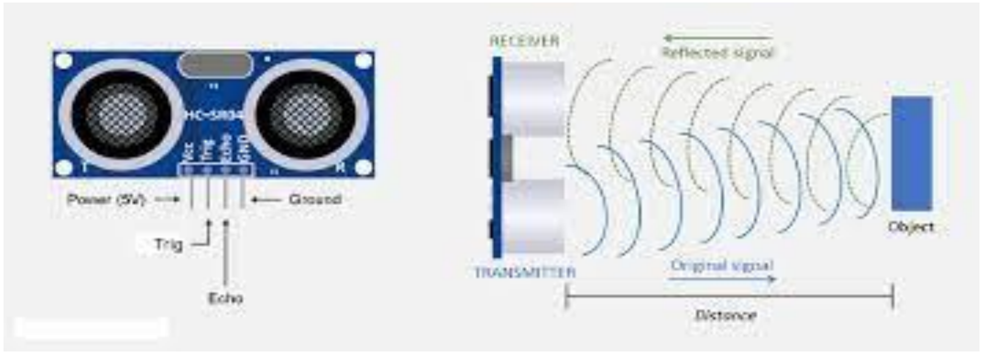


Fig 2 Ultrasonic sensor

3. LIDAR sensor

LiDAR, or light detection and ranging, is a popular remote sensing method used for measuring the exact distance of an object on the earth’s surface. Even though it was first used in the 1960s when laser scanners were mounted to airplanes, LiDAR didn’t get the popularity it deserved until twenty years later. It was only during the 1980s after the introduction of GPS that it became a popular method for calculating accurate geospatial measurements. Now that its scope has spread across numerous fields, we should know more about LiDAR mapping technology and how it works.

Fig 3 What is LIDAR sensor

Fig 4 LIDAR Sensor

4. Raspberry Pi Camera

The Raspberry Pi Camera Module v2 replaced the original Camera Module in April 2016. The v2 Camera Module has a Sony IMX219 8-megapixel sensor (compared to the 5-megapixel OmniVision OV5647 sensor of the original camera). The Camera Module can be used to take high-definition video, as well as stills photographs.

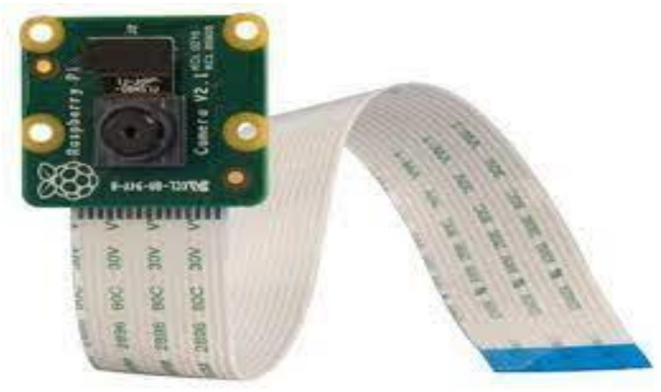


Fig 5 Raspberry Pi Camera

5. DC Geared Motors

A gear motor is an all-in-one combination of a motor and gearbox. The addition of a gear head to a motor reduces the speed while increasing the torque output. Most of our DC motors can be complimented with one of our unique gearheads, providing you with a highly efficient gear motor solution.

A geared DC Motor has a gear assembly attached to the motor. The speed of motor is counted in terms of rotations of the shaft per minute and is termed as RPM . The gear assembly helps in increasing the torque and reducing the speed.

Fig 6 DC Geared Motors

6. L298N Motor Driver

This L298N Motor Driver Module is a high power motor driver module for driving DC and Stepper Motors. This module consists of an L298 motor driver IC and a 78M05 5V regulator. L298N Module can control up to 4 DC motors, or 2 DC motors with directional and speed control.

ENA: Enables PWM signal for Motor A

ENB: Enables PWM signal for Motor B

GND: Ground pin

Pin Name: Description

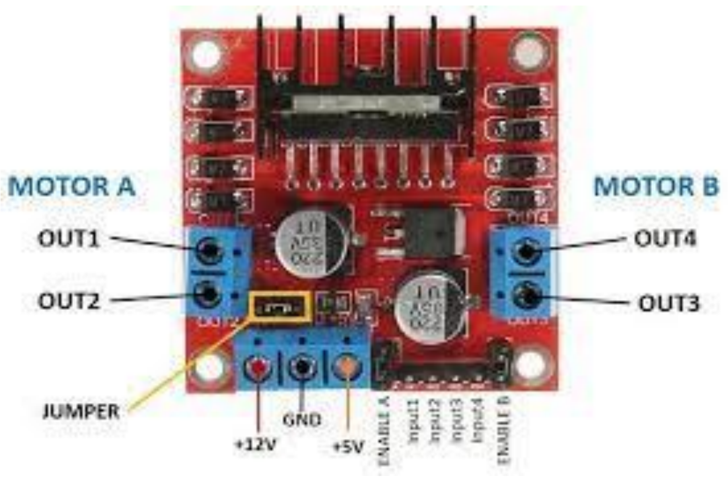


Fig 7 L298N Motor Driver

1.6 Scope

Technological advances are pacing in an unmanageable manner. The more a nation invests in upcoming technologies, the more incentives it enjoys in the long run. These incentives branch from finer infrastructure to augmentation of the jobs sector. The prime focus of technology nowadays is to moderate the human intervention in day-to-day tasks. Machine Learning, Artificial Intelligence, and Computer Vision being the big players in doing so. We have reached a time wherein we can automate our menial tasks like driving. Insane, right? This segment of Computer Vision is popularly known as Autonomous Cars. These are vehicles that perceive their environment and move without human navigation.

Chapter 2

REQUIREMENT SPECIFICATION

2.1 Methods used for Requirement Analysis

An increasing safety and reducing road accidents, thereby saving lives are one of great interest in the context of Advanced Driver Assistance Systems .We have identified that the other traditional system is rely on sensors and lane detection, image processing method which are used for self-driving car in traditional system are not giving accurate result and because of that accident chances is increasing. So, in this project we have tried to overcome these problems by using IoT and merging two methods(image processing and neural network) for more accurate result based real time monitoring system which update the live image data from raspberry pi camera module sensors to the main module and Raspberry Pi is best suited small processor having frequency 1.2 GHz which collects the data from pi camera and for the collection of data we use RPi. GPIO and port of Raspberry Pi having data processing speed of 10Mbps which collect the data and sent to the out main module. So the Collect image data is send as input to our lane module and from lane module we get curve as output, which is then sent to our motor module for handling the speed of motor.

2.2 Data Requirements

We need a data to train our image processing model, so we have created the image dataset on real time using Pi camera and converting colour image to grayscale using thresholding. Also, we need curve value for which we used lane detection which use image dataset and give output as curve value which in used as data in motor module for controlling image.

We also need road sign dataset to train our model, Autonomous vehicles combine a variety of sensors to perceive their surroundings, including radar, lidar, computer vision, sonar, and GPS, among others. These sensors interpret sensory information to identity navigation paths, avoid obstacles and read relevant markers, like road signs

2.3 Functional Requirements

The functional requirements for a system describe what the system should do. Those requirements depend on the type of software being developed, the expected users of the software. These are statement of services the system should provide, how the system should react to inputs and how the system should behave in different situations.

- Image Processing has to be done only once and works automatically on New Realtime Data
- On large data set along with embedded sensors the system works perfectly (Compatibility of Sensors and Software)
- It is platform independent and with some minor changes can run on any OS and with any language.
- Processing speed and memory of the processor and or every module should be good so that data can be easily catch and sent to main module for processing.

2.4 Non-Functional Requirements

1. Performance Requirements

- Robustness: Vehicle should be robust enough to deal with and act accord to changes in road and tire conditions without losing too much performance.
- Quickness: Vehicle's embedded system should be fast enough to interact with the cloud and exchange data with it on the go while responding to the user actions without any shattering or buffering.
- Failure Handling: In case of failures due to unavoidable reasons, the vehicle should be able to recover

2. Safety Requirements
    - Systems Safety: Vehicle will be tested in different environments to make sure it responds safely to software malfunctions, near crashes, near breakdowns, loss of traction and other risks.
    - Ethical Considerations: Humans have to take instant ethical decisions several times when on road. The car will also be programmed to make ethical decisions in case of emergency
    - Detection and Response: Vehicle will be able to detect and respond to other vehicles, pedestrians, animals and traffic signals. It will be able to change lanes, take turns and overtake other vehicles on the road based on the detection.
    - Laws and Practices: Vehicle will follow laws of the location it is being operated in. It will be able to recognize different speed limits and traffic rules of different states and cities. The vehicle should be able to violate the law to avoid any fatal accident.

3. Security Requirements
    - Data Sharing: Self-driving car collects a lot of data on the go. Data and statistics storage will be done to maintain the correct functioning of the car and to reconstruct what went wrong in case of a breakdown
    - Digital Security: Vehicle will be engineered to prevent online threats. All communications between Raspberry Pi and the GCP will be encrypted using SSL

4. Software Quality Attributes
    - Compatibility: Any additional feature that will be added to the self-driving car should peacefully co-exist with existing features
    - Error Handling: Vehicle should not cause or trigger any events that create an accident-prone situation on the road under events like loss of network

2.5 System Specification

2.5.1 Hardware Specification

1. Raspberry pi 4 B model:
    - Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
    - 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)

- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.0 graphics
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0 – 50 degrees C ambient
- A good quality 2.5A power supply can be used if downstream USB peripherals consume less than 500mA in total.

- Raspberry pi camera sensor

2. Motor Control System

- DC Geared Motors
- L298N Motor Driver
- Power booster
- Ac to Dc converter
- Motor driver controller

2.5.2 Software Specification

- Python
- Linux OS
- MU Code
- TensorFlow lib

Chapter 3

ANALYSIS AND MODELLING

3.1 Use Case Model

A Use Case Diagram consists of set of elements and the relationships between them. It depicts all the scenarios, regarding how our application interacts with users and other external systems to achieve the goals of application. The main components of a use case diagram include actors, use cases and their relationships. The use case is an external view of the system that represents some actions that the user performs to get a job done. Actors are the users who interact with the application.

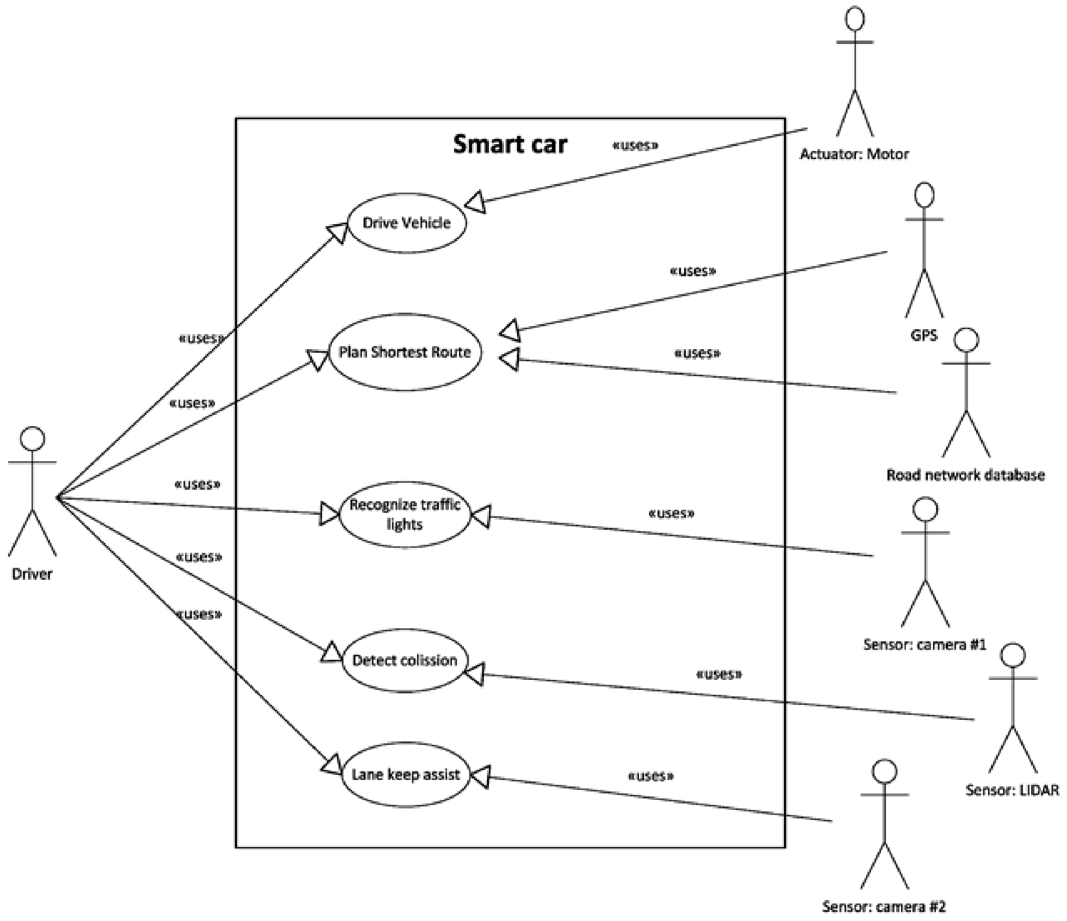


Fig 8 – Use Case Model

3.2 Class Diagram

A class diagram is type of structure diagram that describes the structure of system by showing the systems, their attributes , operations and the relationships among objects.

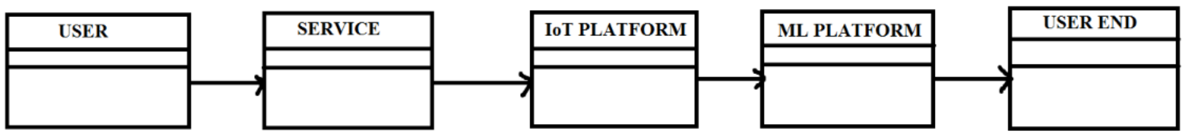
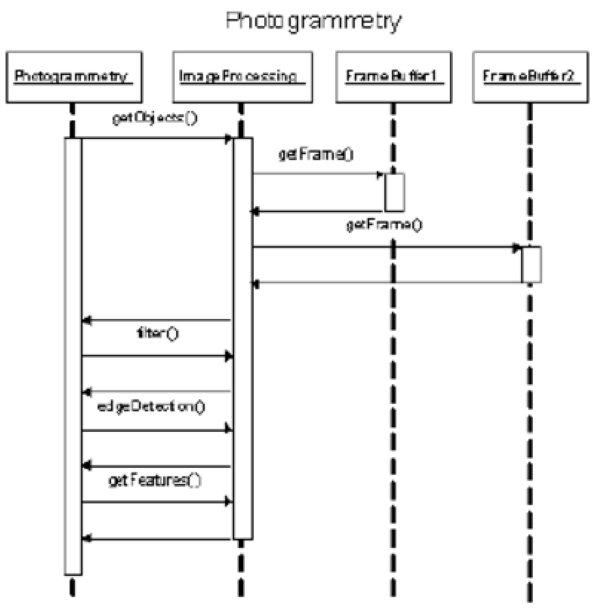


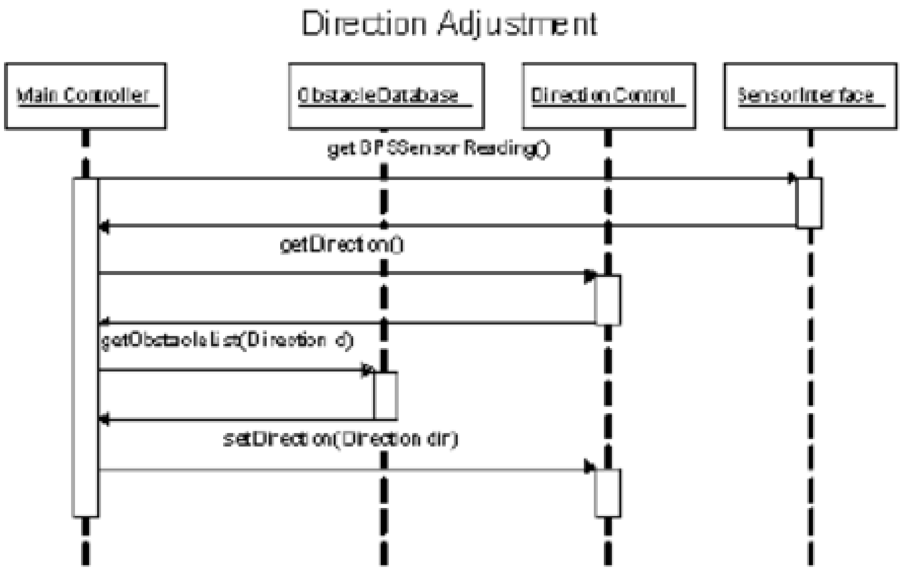
Fig 9 – Class Diagram

3.3 Interaction Diagrams

3.3.1 Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function.





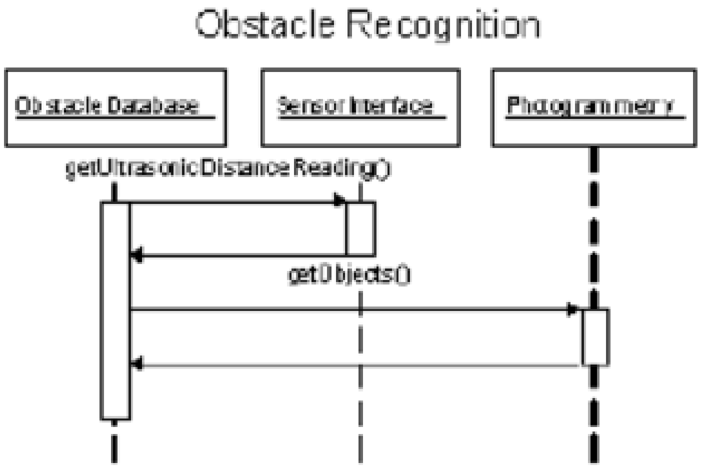


Fig 10 – Sequence Diagram

3.3.2 Collaboration Diagram

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). These diagrams can be used to portray the dynamic behavior of a particular use case and define the role of each object.

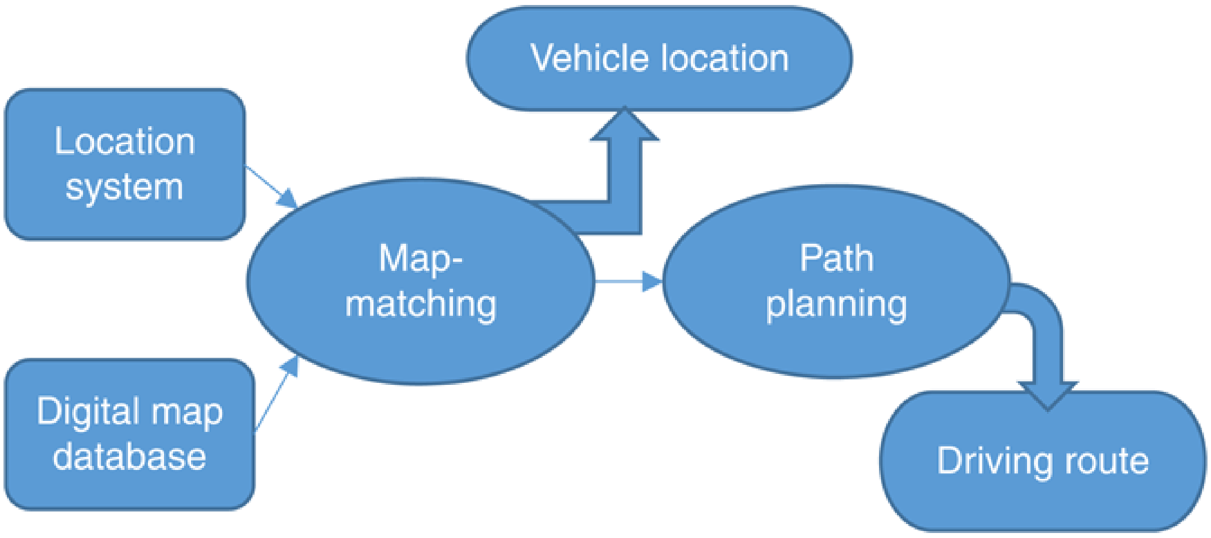


Fig 11 – Collaboration Diagram

3.4 Activity Diagram

An activity diagram is a behavioral diagram i.e. it depicts the behavior of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.

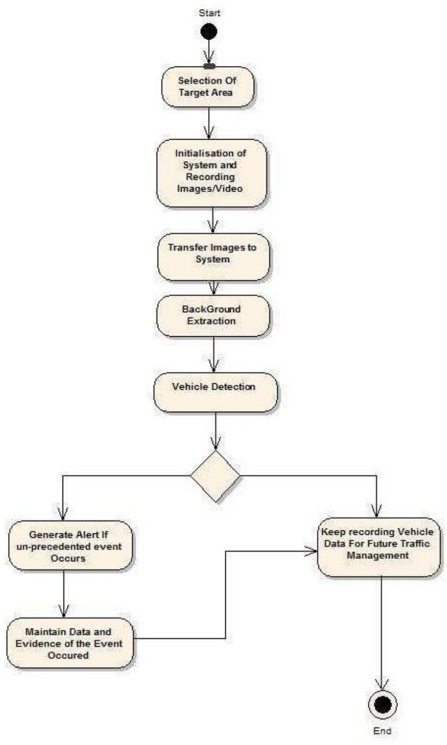

Start
Selection Of Target Area
Initialisation of System and Recording Images/Video
Transfer Images to System
BackGround Extraction
Vehicle Detection
Generate Alert If un-precedented event Occurs
Keep recording Vehicle Data For Future Traffic Management
Maintain Data and Evidence of the Event Occured
End

Fig 12 – Activity Diagram

3.5 Deployment Diagram

A deployment diagram in the Unified Modeling Language models the physical deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components exist, what software components run on each node, and how the different pieces are connected

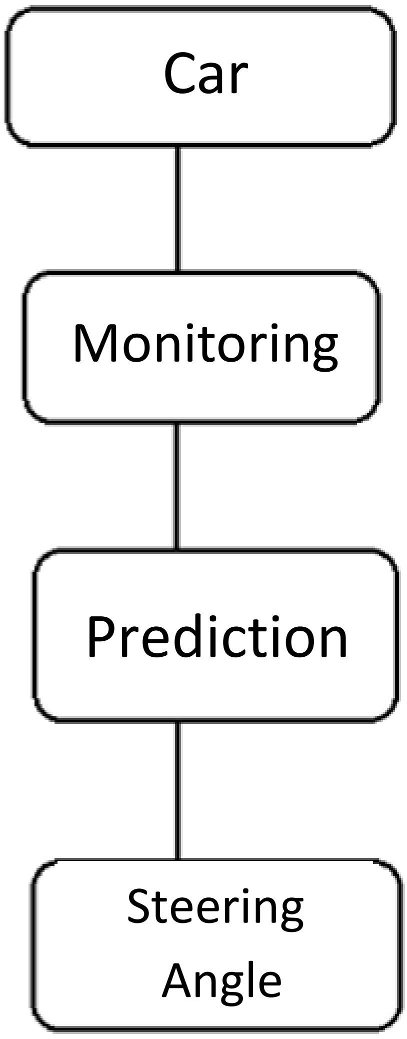


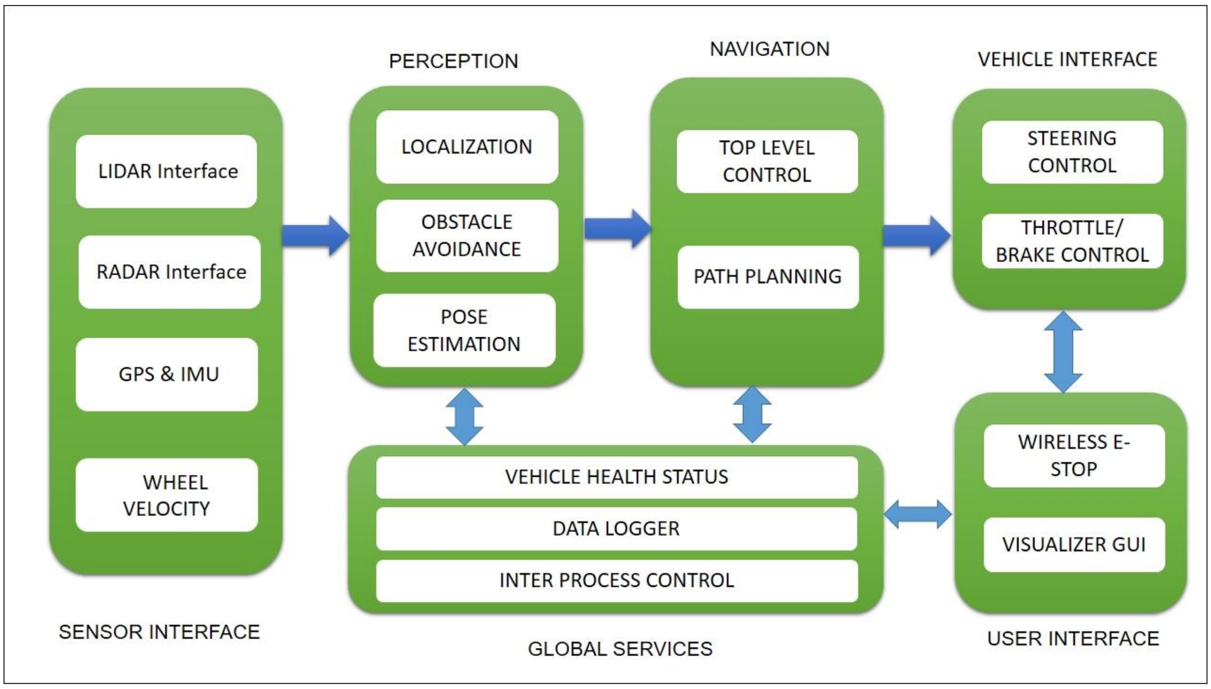
Fig 13 – Deployment Diagram

Chapter 4

SYSTEM DESIGN

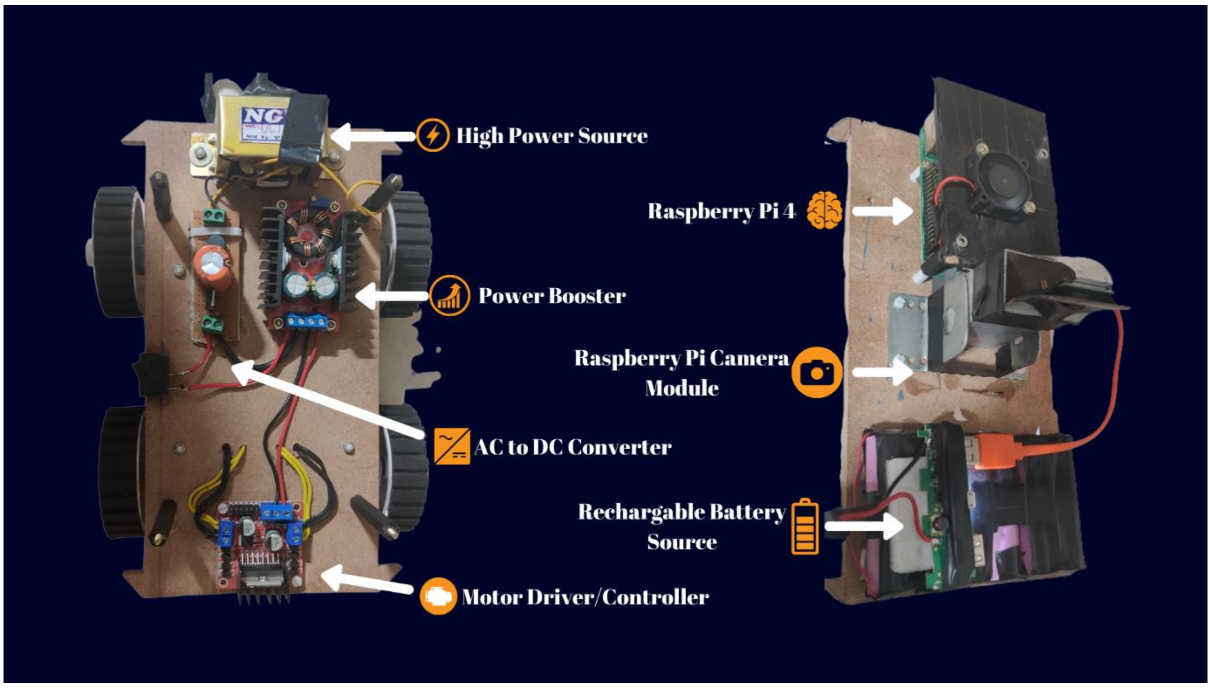
4.1 User Interface Design

Since this an autonomous self-driving car, the user does not directly interact with the car. Rather, the car may have some in-dash computer system, like those found in modern day cars. The in-dash systems, apart from providing entertainment features and vital information like speed and engine RPM, provide navigation aids like GPS as well. They, thus, act as an interface between the user and the car. Such in-dash systems and modifications required for them to work with our system, however, are beyond the scope of this project.



4.2 System Architecture

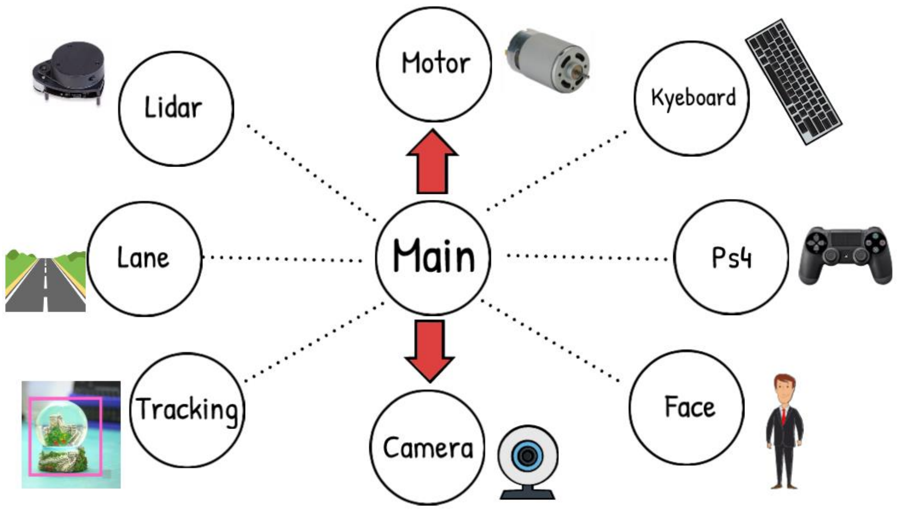
This is the architecture of our project that we have used. It consists of a power source made up of a 3 Ampere transformer and an Ac to Dc Converter. It also has a Booster to give maximum voltage to the motors. The motors are connected through the motor driver that drives the motors. The brain of the system is a 4GB Ram Raspberry Pi 4 Model that has inbuilt Wi-Fi and General-purpose input output pins. A camera module compatible with raspberry Pi is connected to receive the live images. A lithium-Ion battery pack is used to power the Raspberry pi separately.

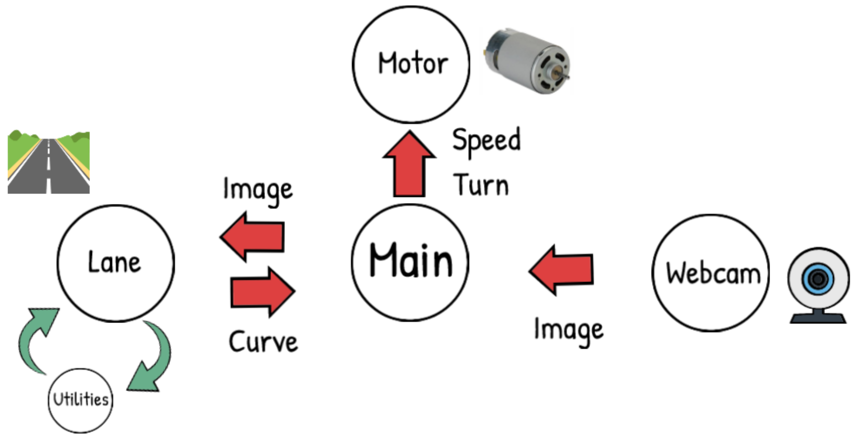


The Concept of Lane Detection

The idea is to find the path using colour detection or edge detector and then getting the curve using summation of pixels in the y direction i.e., a histogram. Splitting the task into 5 different steps. This includes Thresholding, Warping, Histogram, Averaging, and Displaying. Since we have been creating modules so far, we are going to create a module for lane detection as well. This way we don't need to put all the code in one script instead we can have separate python files that each perform their separate tasks. So, for this project we will have a Main Script that will be connected to our Motor Module and the Lane Detection

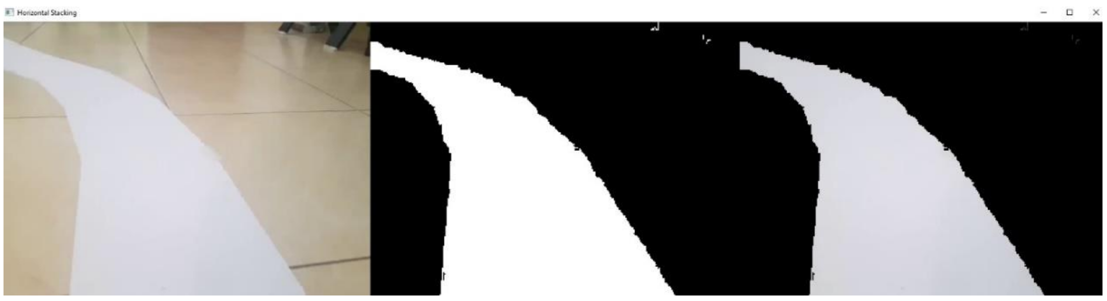
Module. Since the Lane Detection code will take up some space, we will separate all the functions into a Utilities file keeping the main Module neat and clean.





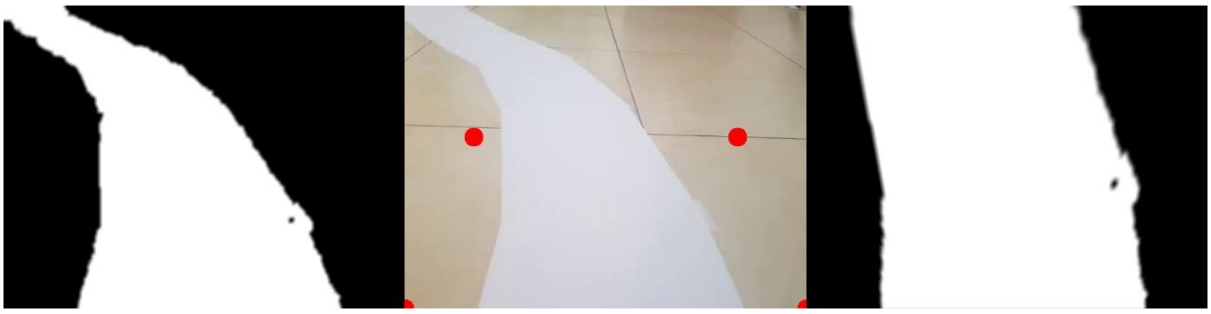
STEP 1 – Thresholding

The idea here is to get the path using either Colour or Edge Detection. Converting the image to HSV colour space and then applying a range of colour to find.



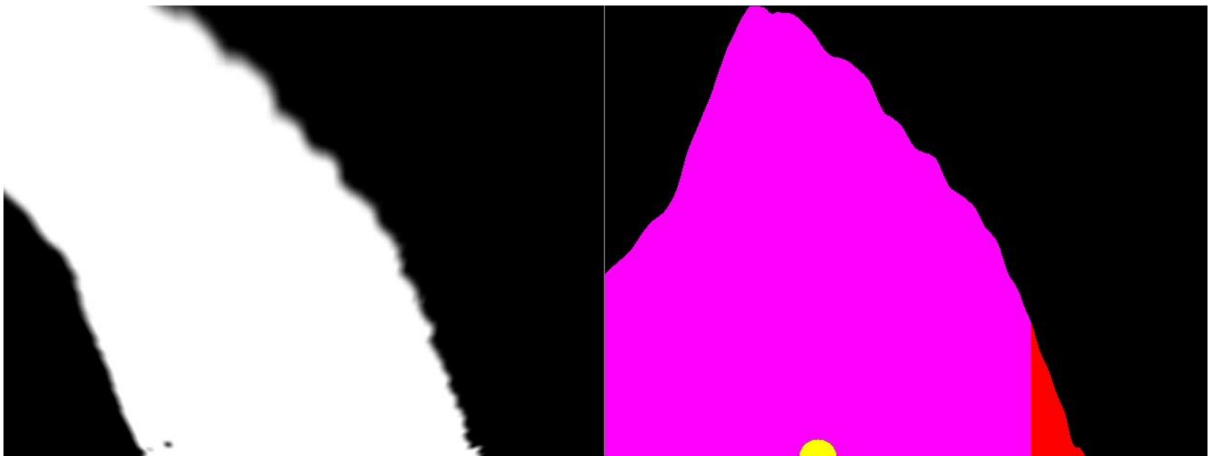
STEP 2- Wrapping

Instead of processing the whole image we just want to know the curve on the path right now and not a few seconds ahead. So, we can simply crop our image, but this is not enough since we want to look at the road as if we were watching from the top. This is known as a bird eye view and it is important because it will allow us to easily find the curve. To warp the image, we need to define the initial points. These points we can determine manually. So, to make this process easier we could use track bars to experiment with different values. The idea is to get a rectangle shape when the road is straight.



STEP 3 - Histogram

Now comes the most important part, finding the curve in our path. To do this we will use the summation of pixels. Given that our Warped image is now binary i.e. it has either black or white pixels, we can sum the pixel values in the y direction. Let's look at this in more detail.

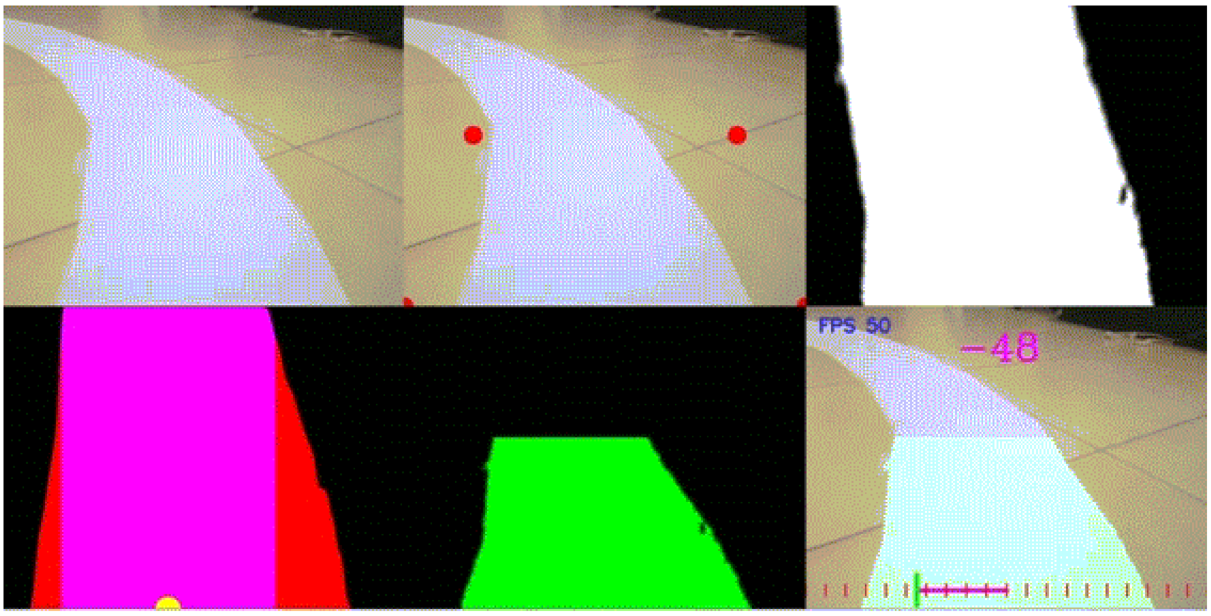


STEP 4 – Averaging

Once we have the curve value, we will append it in a list so that we can average this value. Averaging will allow smooth motion and will avoid any dramatic movements.

STEP 5 – Display

We can add options to display the final result. We will add an input argument to our main function so that we can have the flexibility of turning it on and off, since raspberry pi would run at very slow speeds if we display and run at the same time.



4.4 Algorithm or pseudo code

Main.py

```
import cv2
import numpy as np
from tensorflow.keras.models import load_model

import WebcamModule as wM
import MotorModule as mM

########################################
steeringSen = 0.70  # Steering Sensitivity
maxThrottle = 0.22  # Forward Speed %
motor = mM.Motor(2, 3, 4, 17, 22, 27) # Pin Numbers
model = load_model('/home/pi/Desktop/My Files/RpiRobot/model_V1.h5')
#######################################

def preProcess(img):
    img = img[54:120, :, :]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img, (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img / 255
    return img

while True:

    img = wM.getImg(True, size=[240, 120])
    img = np.asarray(img)
    img = preProcess(img)
    img = np.array([img])
    steering = float(model.predict(img))
    print(steering*steeringSen)
    motor.move(maxThrottle,-steering*steeringSen)
    cv2.waitKey(1)
```

Training.py

```
print ('Setting UP')
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
```

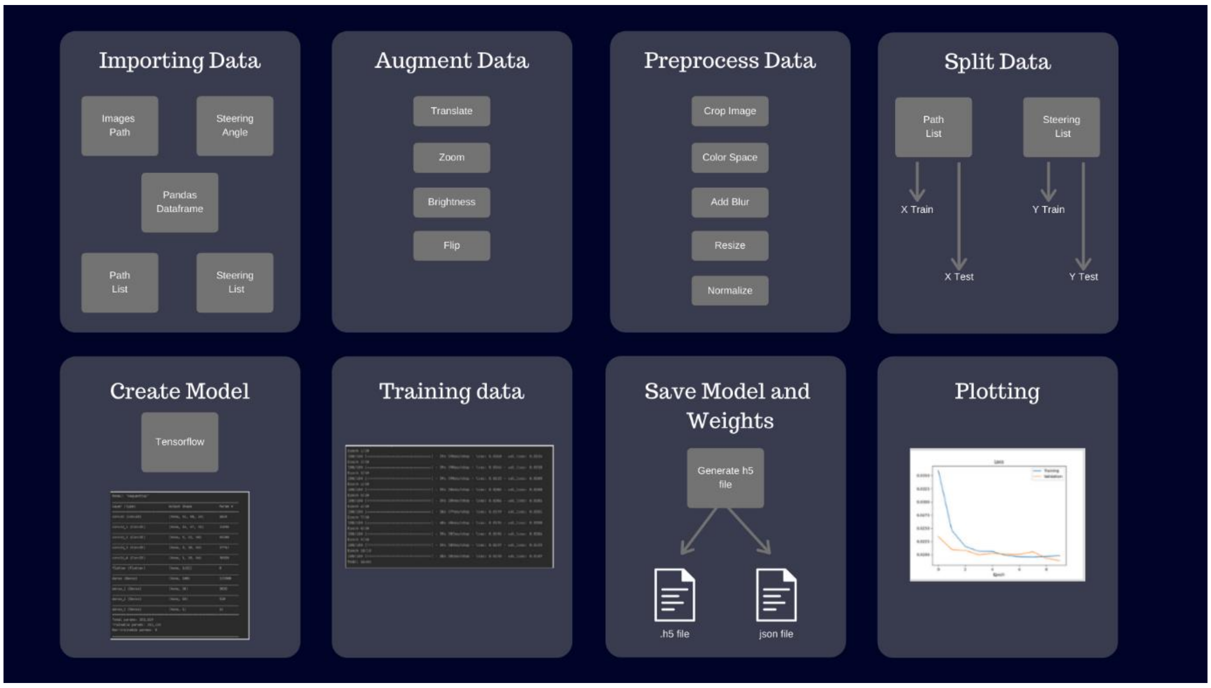
```
from sklearn.model_selection import train_test_split
from utlis import *



#### STEP 1 - INITIALIZE DATA
path = 'DataCollected'
data = importDataInfo(path)
print(data.head())

#### STEP 2 - VISUALIZE AND BALANCE DATA
data = balanceData(data,display=True)

#### STEP 3 - PREPARE FOR PROCESSING
imagesPath, steerings = loadData(path,data)
# print('No of Path Created for Images ',len(imagesPath),len(steerings))
# cv2.imshow('Test Image',cv2.imread(imagesPath[5]))
# cv2.waitKey(0)

#### STEP 4 - SPLIT FOR TRAINING AND VALIDATION
xTrain, xVal, yTrain, yVal = train_test_split(imagesPath, steerings,
                                              test_size=0.2,random_state=10)
print('Total Training Images: ',len(xTrain))
print('Total Validation Images: ',len(xVal))

#### STEP 5 - AUGMENT DATA
#### STEP 6 - PREPROCESS

#### STEP 7 - CREATE MODEL
model = createModel()

#### STEP 8 - TRAINNING
history = model.fit(dataGen(xTrain, yTrain, 100, 1),
                              steps_per_epoch=100,
                    epochs=10,
                    validation_data=dataGen(xVal, yVal, 50, 0),
                    validation_steps=50)

#### STEP 9 - SAVE THE MODEL
model.save('model.h5')
print('Model Saved')

#### STEP 10 - PLOT THE RESULTS
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Training', 'Validation'])
plt.title('Loss')
plt.xlabel('Epoch')
plt.show()
```

utils.py

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
import cv2

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D,Flatten,Dense
from tensorflow.keras.optimizers import Adam

import matplotlib.image as mpimg
from imgaug import augmenters as iaa

import random
```

```
#### STEP 1 - INITIALIZE DATA
def getName(filePath):
    myImagePathL = filePath.split('/')[-2:]
    myImagePath = os.path.join(myImagePathL[0],myImagePathL[1])
    return myImagePath

def importDataInfo(path):
    columns = ['Center','Steering']
    noOfFolders = len(os.listdir(path))//2
    data = pd.DataFrame()
    for x in range(17,22):
        dataNew = pd.read_csv(os.path.join(path, f'log_{x}.csv'), names = columns)
        print(f'{x}:{dataNew.shape[0]} ',end='')
        #### REMOVE FILE PATH AND GET ONLY FILE NAME
        #print(getName(data['center'][0]))
        dataNew['Center']=dataNew['Center'].apply(getName)
        data =data.append(dataNew,True )
    print(' ')
    print('Total Images Imported',data.shape[0])
    return data

#### STEP 2 - VISUALIZE AND BALANCE DATA
def balanceData(data,display=True):
    nBin = 31
    samplesPerBin =  300
    hist, bins = np.histogram(data['Steering'], nBin)
    if display:
        center = (bins[:-1] + bins[1:]) * 0.5
        plt.bar(center, hist, width=0.03)
        plt.plot((np.min(data['Steering']), np.max(data['Steering'])), (samplesPerBin,
samplesPerBin))
        plt.title('Data Visualisation')
        plt.xlabel('Steering Angle')
        plt.ylabel('No of Samples')
        plt.show()
    removeindexList = []
    for j in range(nBin):
        binDataList = []
        for i in range(len(data['Steering'])):
            if data['Steering'][i] >= bins[j] and data['Steering'][i] <= bins[j + 1]:
                binDataList.append(i)
        binDataList = shuffle(binDataList)
        binDataList = binDataList[samplesPerBin:]
        removeindexList.extend(binDataList)

    print('Removed Images:', len(removeindexList))
    data.drop(data.index[removeindexList], inplace=True)
    print('Remaining Images:', len(data))
    if display:
        hist, _ = np.histogram(data['Steering'], (nBin))
        plt.bar(center, hist, width=0.03)
        plt.plot((np.min(data['Steering']), np.max(data['Steering'])), (samplesPerBin,
samplesPerBin))
        plt.title('Balanced Data')
        plt.xlabel('Steering Angle')
        plt.ylabel('No of Samples')
        plt.show()
    return data

#### STEP 3 - PREPARE FOR PROCESSING
def loadData(path, data):
  imagesPath = []
  steering = []
  for i in range(len(data)):
    indexed_data = data.iloc[i]
    imagesPath.append( os.path.join(path,indexed_data[0]))
    steering.append(float(indexed_data[1]))
  imagesPath = np.asarray(imagesPath)
  steering = np.asarray(steering)
  return imagesPath, steering


#### STEP 5 - AUGMENT DATA
def augmentImage(imgPath,steering):
    img =  mpimg.imread(imgPath)
    if np.random.rand() < 0.5:
        pan = iaa.Affine(translate_percent={"x": (-0.1, 0.1), "y": (-0.1, 0.1)})
        img = pan.augment_image(img)
    if np.random.rand() < 0.5:
        zoom = iaa.Affine(scale=(1, 1.2))
        img = zoom.augment_image(img)
    if np.random.rand() < 0.5:
        brightness = iaa.Multiply((0.5, 1.2))
        img = brightness.augment_image(img)
    if np.random.rand() < 0.5:
        img = cv2.flip(img, 1)
        steering = -steering
    return img, steering

# imgRe,st = augmentImage('DataCollected/IMG18/Image_1601839810289305.jpg',0)
# #mpimg.imsave('Result.jpg',imgRe)
# plt.imshow(imgRe)
# plt.show()

#### STEP 6 - PREPROCESS
def preProcess(img):
    img = img[54:120,:,:]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img,  (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img/255
    return img

# imgRe =
preProcess(mpimg.imread('DataCollected/IMG18/Image_1601839810289305.jpg'))
# # mpimg.imsave('Result.jpg',imgRe)
# plt.imshow(imgRe)
# plt.show()

#### STEP 7 - CREATE MODEL
def createModel():
  model = Sequential()

  model.add(Convolution2D(24, (5, 5), (2, 2), input_shape=(66, 200, 3), activation='elu'))
  model.add(Convolution2D(36, (5, 5), (2, 2), activation='elu'))
  model.add(Convolution2D(48, (5, 5), (2, 2), activation='elu'))
  model.add(Convolution2D(64, (3, 3), activation='elu'))
  model.add(Convolution2D(64, (3, 3), activation='elu'))

  model.add(Flatten())
  model.add(Dense(100, activation = 'elu'))
  model.add(Dense(50, activation = 'elu'))
  model.add(Dense(10, activation = 'elu'))
  model.add(Dense(1))

  model.compile(Adam(lr=0.0001),loss='mse')
  return model

#### STEP 8 - TRAINNING
def dataGen(imagesPath, steeringList, batchSize, trainFlag):
    while True:
        imgBatch = []
```

```
steeringBatch = []

for i in range(batchSize):
    index = random.randint(0, len(imagesPath) - 1)
    if trainFlag:
        img, steering = augmentImage(imagesPath[index], steeringList[index])
    else:
        img = mpimg.imread(imagesPath[index])
        steering = steeringList[index]
    img = preProcess(img)
    imgBatch.append(img)
    steeringBatch.append(steering)
yield (np.asarray(imgBatch),np.asarray(steeringBatch))
```



Chapter 5

TESTING

5.1 Testing Objectives

Software testing is a critical element of software quality assurance and represents the ultimate service of specification design and coding. The increasing visibility of software as a system element and the attended costs associated with the software failure and motivating forces for well planned, thorough testing. It is not unusual for a software developer to spend between 30 and 40 percent of total project effort in testing. System Testing Strategies for this system integrate test case design techniques into a well-planned series of steps that result in the successful construction of this software. It also provides a road map for the developer, the quality assurance organization and the customer, a roadmap that describes the steps to be conducted as path of testing, when these steps are planned and then undertaken and how much effort, time and resources will be required.

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet-undiscovered error. A successful test is one that uncovers a yet undiscovered error. The above objectives imply a dramatic change in viewpoint. They move counter to the commonly held view that a successful test is one in which no errors are found. Our objective is to design tests that systematically verify different clauses of errors and do so with a minimum amount of time and effort. If testing is conducted successfully, it will uncover errors in the software. As a secondary benefit, testing demonstrates that software functions appear to be working according to specification and that performance requirements appear to have been met. In addition, data collected as testing is conducted provides a good indication of software. Testing can’t show the absence of defects, it can only show that software errors are present. It is important to keep this in mind as testing is being conducted.

Testing Principles:

Before applying methods to design effective test cases, a software engineer must understand the basic principles that guide software testing.

- All tests should be traceable to customer requirements.
- Tests should be planned long before testing begins.
- Testing should begin "in the small" and progress towards testing "in the large"
- Exhaustive testing is not possible.

Test Plan:

A test plan is a document that contains a complete set of test cases for a system, along with other information about the testing process. The test plan should be returned long before the testing starts.

Test plan identifies:

- A task set to be applied as testing commences,
- The work products to be produced as each testing task is executed
- The manner, in which the results of testing are evaluated, recorded and reuse when regression testing is conducted.

In some cases, the test plan is indicated with the project plan. In others, the test plan is a separate document. The test report is a record of the testing performed. The testing report enables the acquirer to assess the testing and its results. The test report is a record of the testing performed. The testing report enables the acquirer to assess the testing and its results.

5.2 Testing Methods and Strategies Used

The testing methods and strategies used for testing the program are as follows:

System testing

- **Software Testing**

  As the coding is completed according to the requirement, we must test the quality of the software. Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding.

  Although testing is to uncover the errors in the software it also demonstrates that software functions appear to be working as per the specifications, those performance requirements appear to have been met. In addition, data collected as testing is conducted provide a good indication of software and some indications of software quality.

To assure the software quality we conduct both White Box Testing and Black Box Testing

- **White Box Testing**

  White Box Testing is a test case design method that uses the control structure of the procedural design to derive test cases. As we are using a non-procedural language, there is a very small scope for the White Box Testing.

  Whenever it is necessary, the control structure is tested and successfully passed all the control structure with a very minimum error.

- **Black Box Testing**

Black Box Testing focuses on the functional requirements of the software. It enables to derive sets of input conditions that will fully exercise all functional requirements for a program. The Black Box Testing finds almost all errors. If finds some interface errors and errors in accessing the database and some performance errors.

- **Code Testing**

Specification testing is done to check if the program does with it should do and how it should behave under various conditions or combinations and submitted for processing in the system and it's checked if any overlaps occur during the processing. This strategy examines the logic of the program. Here only syntax of the code is tested. In code testing syntax errors are corrected, to ensure that the code is perfect.

- **Unit Testing**

The first level of testing is called unit testing. Here different modules are tested against the specifications produced during the design of the modules. Unit testing is done to test the working of individual modules with test oracles.

Unit testing comprises a set of tests performed by an individual programmer prior to integration of the units into a large system. A program unit is small enough that the programmer who developed if can test it in detail.

Unit testing focuses first on the modules to locate errors. These errors are verified and corrected so that the unit perfectly fits to the project.

- **System Testing**

The next level of testing is system testing and acceptance testing. This testing is done to check if the system has its requirements and to find the external behavior of the system. System testing involves two kinds of activities:

  - **Integration Testing**

Integration testing is the level of software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before system testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

Bottom-up Integration

This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.

Top-down Integration

In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

- **Acceptance Testing**

This testing is performed finally by user to demonstrate that the implemented system satisfies its requirements. The user gives various inputs to get the required outputs.

- **System Testing**

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black-box testing, and as such, should require no knowledge of the inner design of the code or logic System testing takes as its input an integrated software system that has passed integration testing phase. System testing is usually considered appropriate for assessing the nonfunctional system requirements.

- **Hardware Testing**

A typical hardware testing process

1. Create a testing environment (e.g., measurement hardware, test software, cabling, fixtures, etc.)
2. Place part into the condition needed for the measurement (apply pressure, voltage, temperature, etc.)
3. Take some measurements
4. Put those measurements through one or more pass/fail criteria
5. Record the results as either summary data or verbose raw plus summary data
6. Repeat 2-5 as needed to sweep through input conditions
7. Create a final report document
8. Declare the part as good or bad
9. Repeat 2-8 for as many parts as need to be tested

Some common components/sub-assemblies to test include:

For electronic parts:

- power supply voltages and currents,
- signal levels and frequencies at various test points,
- range of operation to check linearity and accuracy,
- and so on.

For mechanical parts:

- dimensional tolerances,
- range of motion (i.e., speed, distance),
- forces,
- temperatures,
- power draw and output for efficiency measurements,
- flow rates,
- and on and on, because there are so many types of mechanical parts.

For optical parts:

- mechanical tolerances,
- power input and output,
- transmission and reflection properties as a function of wavelength,
- and so on.

For communications parts:

- bandwidths,
- transmission power,
- receive power,
- bit-error-rates,
- distortion,
- and so on.

5.3 Test Case

A TEST CASE is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly. The process of developing test cases can also help find problems in the requirements or design of an application.

A test case is usually a single step, or occasionally a sequence of steps, to test the correct behaviour/functionality, features of an application. An expected result or expected outcome is usually given.
Typical Test Case Parameters:

- Test Case ID
- Test Scenario
- Test Case Description
- Test Data
- Expected Result
- Test Parameters
- Actual Result
- Table 1: Test Cases

TEST CASE NUMBER	TEST CASE	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	RESULT

1	Camera Module	Image/Video	Displays video /image	Displays given Image / Video	PASS
2	Motor Module	Give value to move Forward/Backward, Turn, Delay	Car moves forward/backward Turn left/Right	Car moves forward/backward Turn left/Right	PASS
3	Warping	Image to be Warped	Warped Image	Warped Image	PASS
4.	Neural Networks	Image of path	Steering Angle	Steering Angle	PASS

5.4 Test Validation

The system has been tested and implemented successfully and thus ensured that all the requirements as listed in the software requirements specification are completely fulfilled. In case of erroneous input corresponding error messages are displayed.

Validation testing is the process of ensuring if the tested and developed software satisfies the client /user needs. The business requirement logic or scenarios must be tested in detail. All the critical functionalities of an application must be tested here

As a tester, it is always important to know how to verify the business logic or scenarios that are given to you. One such method that helps in detail evaluation of the functionalities is the Validation Process.

Whenever you are asked to perform a validation test, it takes a great responsibility as you need to test all the critical business requirements based on the user needs. There should not be even a single miss on the requirements asked by the user. Hence a keen knowledge on validation testing is much important.

Chapter 6

CONCLUSION AND FUTURE WORK

a. Limitations Of Project

1. Expensive

Self-driving cars are so exciting because they are stuffed to the brim with space age technology, but all this technology is currently astronomically expensive. In general, technology grows cheaper the longer it is available to the public, so self-driving cars may eventually be something anyone can afford. For now, however, most companies have not released a price for their driverless cars.

2. Potential For Technology To Go Wrong

Though successful programming lets us do incredible things, there is always the potential for some unexpected glitch to emerge. Even if a self-driving car performs flawlessly at first, it is possible for the programming that runs the cars to be updated by the car company with a fault string of code. Errors like this cause annoyance on our computers and mobile devices, but could potentially cause car accidents with self-driving cars.

3. Prone to Hacking

Autonomous vehicles could be the next major target of the hackers as this vehicle continuously tracks and monitors details of the owner. This may lead to the possible collection of personal data.

4. Non-functional sensors

Sensor failures often happened during drastic weather conditions. This may not work during a blizzard or a heavy snowfall.

1.2 Future Enhancements

1. Research has shown that the number of U.S. deaths resulting from road accidents could be reduced by more than 90% by the year 2050 because of self-driving cars.
2. More Independence for the Physically Challenged
3. Today's cars are not user-friendly for the elderly or disabled, or anyone else who may have difficulty getting around. Paratransit services with wheelchair accommodation—or even hospital bed accommodation—and cars with Braille buttons will be normal.
4. Reduced Emissions
5. Driverless cars can reduce harmful emissions by up to 60%. Furthermore, these cars can be programmed to maximize the potential reductions, which is amazing news for the environmentally conscious and anyone who wants to leave a minimum impact on Mother Earth.

1.3 Conclusion

Self-driving cars are autonomous vehicles that can drive by themselves without any human interference and has the potential to mark the technological revolution of the next decade. This work presents the development of a low-cost prototype of a miniature self-driving car model using simple and easily available technologies. The objective of the work is to avoid accidents caused due to driver faults

Technologies such as GPS to access location, sensors for obstacle detection and avoidance, image processing, computer vision for processing images and Neural Network for intelligent systems have been deployed.

Chapter 7

BIOBLIOGRAPHY AND REFERENCES

7.1 Reference Books

- “Arduino Architecture” *https://www.engineersgarage.com/what-is-gsm-gprs-module*
- “Systems design” *https://en.wikipedia.org/wiki/Systems_design*
- “UML - Standard Diagrams” *https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm*
- “The Internet of Things in healthcare: an overview” *https://scholar.google.com/citations?user=Y4opLB8AAAAJ&hl=en*
- “Envisioning inclusive futures: technology-based assistive sensory and action substitution” *https://www.infona.pl/resource/bwmeta1.element.elsevier-3d45bfdd-fe55-359f-84e4-674a21cae024*

7.2 Other Documentations and Resources

- “A multiple communication standards compatible IoT system for medical usage” *http://ieeexplore.ieee.org/document/6577775/?reload=true*
- “Ubiquitous data accessing method in IoT-based information system for emergency medical services” *https://www.deepdyve.com/lp/institute-of-electrical-and-electronics-engineers/ubiquitousdata-accessing-method-in-iot-based-information-system-for-YCZzyY5W9g*
- “Implementation of a medical support system considering P2P and IoT technologies” *https://www.computer.org/csdl/proceedings/cisis/2014/4325/00/4325a101-abs.html*

- “Secure end-to-end communication for constrained devices in IoT-enabled ambient assisted living systems”
  https://www.computer.org/csdl/proceedings/wf-iot/2015/03